REMARKS

Prior to this Amendment, claims 1-6, 8-20 and 22 were pending in the application.

Claims 1, 12, and 20 have been amended to clarify that the claimed invention involves a system and method in which a virtual machine simultaneously receives and concurrently dispatches, manipulates, and processes a number of platform-specific events that are initiated externally to the virtual machine with support found at least in Paragraphs [0024], [0027]-[0029], [0033]-[0035], [0041]-[0042] and Figures 1B and 2 of the original specification. Claim 4 has been amended to correct informalities identified in the claim language. Claims 8-10 have been amended to correct antecedent basis issues identified in the claim language, and claim 13 has been amended to correct antecedent basis issues necessitated by the amendment to claim 12. Claims 14-19 have been canceled.

After entry of the Amendment, claims 1-6, 8-13, 20, and 22 remain for consideration by the Examiner.

Claim Rejections Under 35 U.S.C. §112

Claims 1-6, 8-20 and 22 stand rejected under 35 U.S.C. § 112 as being indefinite for failing to comply with the written description requirement. Specifically, the Examiner asserts that the specification fails to describe (1) a "concurrent determination" regarding which of first and second tasks should process each platform-specific event (i.e., concurrently determining a selected task), and (2) a "manipulation" of each platform-specific event received by modifying a data structure of the platform-specific event to comply with a data structure format supported by the selected task. Applicant disagrees.

The fundamental factual inquiry with respect to the written description requirement is whether the specification conveys with "reasonable clarity" to those skilled in the art that the applicant was in possession of the invention as claimed at the time of filing. MPEP § 2163.02. Further, an applicant may show possession of the

claimed invention using many different descriptive means such as words, structures, figures, diagrams, and formulas. Id.

With respect to a "concurrent determination" of a selected task, or concurrently determining which of first and second tasks should process each platform-specific event, paragraph [0027] explains and Figure 1B clearly shows that several platform-specific events E1, E2, E3, and E4 are simultaneously received by a virtual machine 112. Further, paragraphs [0027]-[0029] reference Figure 1B in explaining that once the virtual machine 112 receives the platform-specific events E1, E2, E3, and E4, a smart event-dispatcher 118 within the virtual machine 112 may determine which one of two concurrent tasks 120, 122 operating on the virtual machine 112 should receive each of the platform-specific events E1, E2, E3, and E4 for processing.

Paragraphs [0024] and [0029] further explain that the smart event-dispatcher 118 may make this concurrent determination based on a set of criteria associated with various attributes of the particular platform-specific event. For example, some events may be pre-assigned to a particular task. In another embodiment, events may be delivered to a task that is currently in the foreground, or being displayed to a user. In yet another embodiment, events may be dispatched to tasks based on prior user selections. Paragraph [0033] further explains that in an embodiment shown in Figure 2, a virtual machine 204 may include a smart event-dispatcher 212 that employs event-dispatching logic 218. The event-dispatching logic 218 concurrently determines, based on the criteria discussed above, which task should receive each of the platform-specific events.

With respect to simultaneously manipulating each of the platform-specific events, paragraph [0034] of the original specification explains that the smart event-dispatcher 212 shown in Figure 2 can manipulate data before placing it in an event repository 220 associated with the selected task. Further detail is provided in Paragraphs [0041]-[0042], which explains that tasks (e.g., tasks 120, 122 of Figure 1B or tasks 214, 216 of Figure 2) that are implemented in Java may not be able to access event data structures that are implemented in another programming language (i.e., a native programming language). In this regard, the virtual machine (e.g., virtual machine 112 of Figure 1B or virtual machine 204 of Figure 2) may be implemented in programming languages other

than Java (e.g., C, C++) such that the code for the virtual machine itself can be used to access data structures of platform-specific events that are implemented in native programming languages. Thus, the virtual machine may manipulate data structures for platform-specific events so as to represent the events as Java data structures. Once manipulated, the Java tasks operating on the virtual machine may access the Java data structures. As a result, the virtual machine itself performs data manipulation, and Java code need not be executed to manipulate data to be Java compliant.

Considering the above, Applicant strongly believes that the specification fully supports the invention as claimed. Thus, Applicant asks that the rejection under 35 U.S.C. § 112 be withdrawn.

Claim Rejections Under 35 U.S.C. § 103 Based on Moore and Nitz

In the Office Action mailed December 18, 2009, claims 1-6, 12-14, 20, and 22 were rejected under 35 U.S.C. § 103(a) as being unpatentable over U.S. Patent No. 7,171,663 ("Moore") in view of U.S. Patent No. 6,370,590 ("Nitz") and further in view of U.S. Patent No. 6,173,332 ("Hickman"). Applicant initially notes that while Hickman is recited in the Examiner's summary of the rejection in paragraph 4 of the Office Action, the Examiner has not actually applied or mentioned Hickman with respect to any of the rejected claims 1-6, 12-14, 20 or 22. Thus, Applicant assumes that the initial mention of Hickman was a typographical error and that the above-noted claims stand rejected as being unpatentable over the combination of Moore and Nitz.

Independent Claim 1

As discussed above, independent claim 1 stands rejected as being unpatentable over the combination of Moore and Nitz. Applicant respectfully requests reconsideration because the proposed combination of Moore and Nitz, even if proper, does not teach each element presented by the combination of features required by claim 1 to one of ordinary skill in the art.

Before detailing the requirements of claim 1, Applicant believes a brief general discussion of the claimed invention along with its unique attributes and benefits may

assist the Examiner in differentiating the claimed invention from the cited references. Fundamentally, the invention involves a system and method of concurrently dispatching each of numerous simultaneously received platform-specific events for processing at one of several tasks that are concurrently operating on a virtual machine. A task is a computing unit that may be represented by a set of programming instructions (e.g., programming instructions written in C, C++, or Java). At task may also be an entire application program, or applet (e.g., a calculator, a music player, a video player, a computer game). See Original Specification, Paragraph [0021].

Operating certain tasks requires the processing of external platform-specific events. That is, certain tasks involve external inputs from users (e.g., an input received from a user via a keyboard or other interface, an input from an external device, etc.). These external inputs, called *external events*, are generally linked to platform-specific hardware or software components (e.g., a specific programming language or operating system), and as result, the external events may be termed "platform-specific" or native events. See Original Specification, Paragraph [0022]. Because tasks operating on a virtual machine are not platform-dependent, the claimed invention provides a mechanism for concurrently assigning and processing several external, platform-specific events that are received simultaneously.

With this in mind, amended claim 1 is directed to a method of processing platform-specific events using a virtual machine that operates on a first platform and that concurrently supports first and second tasks. Claim 1 requires that the virtual machine simultaneously receive a plurality of platform-specific events that are associated with the first platform. Each of the platform-specific events is an *external event* that is initiated externally to the virtual machine. Once the platform-specific events are received, claim 1 requires concurrently determining a selected task for each of the platform-specific events. The selected task defines which of the first and second tasks should process the respective platform-specific event. Claim 1 further requires simultaneously manipulating each of the platform-specific events by modifying a data structure of the event to comply with a data structure format that is supported by the respective selected task. Such modification represents each of the platform-specific events in a form that is accessible by the respective selected task.

As discussed in Paragraphs [0022] and [0027]-[0029] and shown in Figure 1B (below) of the original specification, a virtual machine 112 simultaneously receives several platform-specific events E1, E2, E3, and E4. Each of the platform-specific events E1, E2, E3, and E4 is an external event initiated outside of the virtual machine 112. For example, the platform-specific events may be incoming data received via a network device or interface, user input generated at a keyboard, and so on.

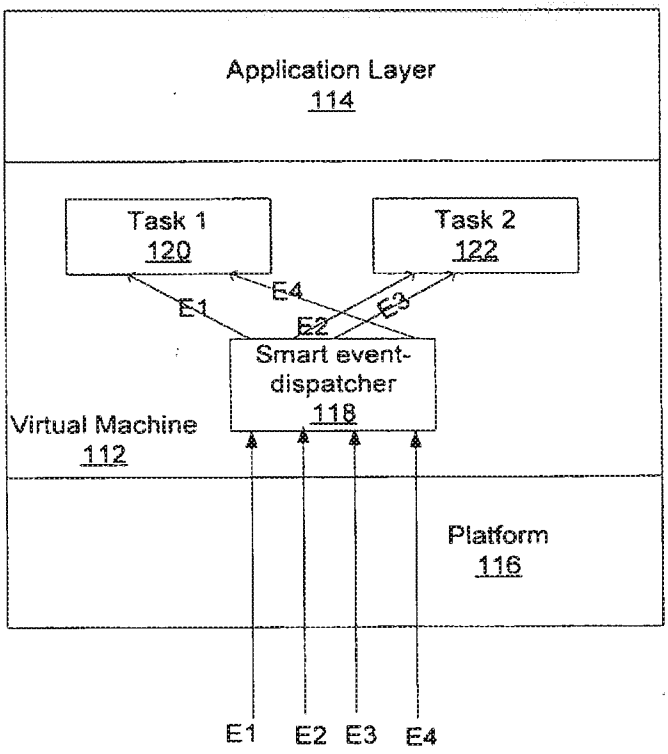

Figure 1B.

A smart event-dispatcher 118 may determine which one of two concurrent tasks 120 and 122 should receive and process each of the external platform-specific events E1, E2, E3, and E4. As discussed above, this concurrent determination regarding how to dispatch each of the external platform-specific events may be based upon a variety of criteria that is implemented within dispatching logic resident on the smart event dispatcher. The criteria may include the entity (i.e., device or interface) through which the event was initiated, which task is running in the foreground, various user-selected or user-programmed parameters, and the like.

Because tasks (e.g., tasks 120, 122 of Figure 1B or tasks 214, 216 of Figure 2) implemented in Java may not be able to access event data structures that are implemented in another (native) programming language, Paragraphs [0041]-[0042] of the original specification explain that the virtual machine (e.g., virtual machine 112 of Figure 1B or virtual machine 204 of Figure 2) may be implemented in programming languages other than Java (e.g., C, C++). In this regard, the code for the virtual machine itself can be used to access platform-specific or native event data structures that are implemented in native programming languages. Thus, the virtual machine may manipulate data structures for platform-specific events so as to represent the events as Java data structures. Once manipulated, the Java tasks may access and process the newly-compatible Java data structures. As a result, the virtual machine itself performs data manipulation, and Java code need not be executed to manipulate data to be Java compliant.

Contrasting the requirements of claim 1, Moore does not teach a virtual machine that *simultaneously* receives a plurality of *external* platform-specific events. Instead, Moore discloses a virtual machine 140 that *serially* receives external events 115, 120, 125. Specifically, Moore teaches two different types of "events" (i.e., external and internal events). Once an external event initiated by a client 105, 110 is received within a virtual machine 140, an event processor 150 may responsively trigger an event dispatcher 160 to broadcast one or more internal VM events. See Moore, column, line 61 – column 5, line 7. It is these internal VM events, not the external events, which are the focus of Moore.

The VM events of Moore are predefined internal events that are derived from consecutively-received external events and that may be detected and processed by event monitors operating on threads within the virtual machine 140. The VM events include internal events such as initializations, destructions, attribute modifications, attribute additions, invalidations, and timeouts. Moore explicitly states that such VM events should be differentiated from external events, which cannot be detected by event monitors within the virtual machine. See Moore, column 4, lines 43-46. Notably, the passages the Examiner has cited as showing receipt of platform-specific events

(column 5, lines 5-7) refer to internal VM events initiated within the virtual machine, not external events as required by claim 1.

In addition, Figure 2 and the corresponding text of Moore clearly show and explain that while internal VM events may be initiated and processed in parallel, external events are received serially such that processing a subsequent external request may interrupt or interfere with processing of a previous request. See Moore, Figure 2; column 6, line 12 – column 7, line 41. That is, Moore discloses a utility for handing external interrupts within a server-side program. The very problem that Moore seeks to address mandates that Moore *receive external events separately*. Specifically, a conventional servlet (e.g., task) within a virtual machine processes external client requests or events in their entirety immediately upon issue from the client. Because the client cannot interrupt or otherwise modify a process executing within the servlet before it is completed, the client must wait until the servlet has completed processing before a subsequent external event may be addressed. See Moore, column 1, lines 30-42. Moore resolves this inefficiency by providing a mechanism for allowing a subsequently initiated external event to interrupt the processing of a previously issued external event. For instance, a client may submit a request that results in a server processing a large Voice Extensible Markup Language document. The server may receive the client request and utilize a servlet to produce a response, but after the servlet has been initiated, an external "hang up" event may occur making the response moot. Moore provides a utility for interrupting the servlet, thereby preventing it from completing the now-meaningless response. See Moore, column 1, lines 43-55.

In sum, to solve the problem that Moore addresses, Moore discloses receiving external requests/events separately such that internal VM events associated with the external requests/events may be allowed to interfere with each other. In contrast, claim 1 requires the virtual machine to receive external events simultaneously, thereby allowing a user and/or network device or interface to initiate several external, platform-specific events or requests at the same time. The simultaneous external events may then be immediately assigned to and processed by an appropriate task within the

platform-independent virtual machine. It is clear that the vastly different purpose of Moore mandates an opposite functionality than that required by the claimed invention.

In addition, the Examiner admits that Moore does not teach the steps of (1) concurrently determining which of the first and second tasks should receive each of the platform-specific events, or (2) simultaneously manipulating each of the platform-specific events to comply with a data structure format supported by the selected task. The Examiner relies on Nitz for the limited purpose of disclosing these steps on a general, but not concurrent, basis. The Examiner further asserts that it would be obvious to one of ordinary skill in the art to combine the teachings of Moore and Nitz and apply the known use of multiple threads to concurrently determine which tasks should receive each of the platform-specific events and simultaneously manipulate each of the platform-specific events. Applicant disagrees.

As discussed above, claim 1 fundamentally involves the concurrent processing of numerous, externally initiated platform-specific events that are simultaneously received by a virtual machine. Nitz discloses a utility for facilitating communication between multiple applications, or sub-applications, that have been integrated to form a larger vertical application. For example, under Nitz, a database application, a spreadsheet application, and a word processing application may be integrated to create a vertical application. Nitz discloses a utility for allowing these three "sub-applications" to communicate with each other. Nitz, Abstract; column 1, lines 13-46; column 4, line 23-column 5, line 8. Nitz is completely silent with respect to determining or selecting which tasks within a virtual machine should receive and process external platform-specific events that the virtual machine has received. Nitz is also silent with respect to manipulating platform-specific events such that they may be accessible by the respective selected tasks. Indeed, Nitz does not involve virtual machines and/or any manner of processing platform-specific events received by a virtual machine.

More specifically, the Examiner cites the Abstract as well as column 7, lines 22-39 and lines 40-55 of Nitz as teaching the steps of concurrently determining which of the first and second tasks should receive each of the platform-specific events and simultaneously manipulating each of the platform-specific events to comply with a data structure format supported by the selected task. A review of these portions of Nitz,

however, reveals a discussion that relates to using an agent/broker architecture to translate and pass messages between sub-applications that form a vertical application. That is, sub-applications integrated within a vertical application may each attach to an agent. Each agent is linked to other agents through a centralized broker that manages message routing between the agents. As sub-applications begin executing, the sub-applications transmit messages to their corresponding agents in the format supported by the respective sub-applications. The agents then translate the messages from the supported format to a common format that all agents understand. The translated massages are transmitted or "published" to the centralized broker, which evaluates and routes the messages to other sub-applications accordingly. While Nitz may involve translating the format of messages passed between computer programs, it simply fails to disclose any type of external event processing via concurrently executing tasks on a virtual machine. Nitz is not remotely related to event processing and is silent with respect to the elements for which it is cited.

In addition, Applicant submits that the proposed combination of Moore and Nitz is improper. In determining whether it would have been obvious to combine the teachings of various references to produce the invention under consideration, a patent examiner must not "read into the prior art the teachings of the invention in issue." Graham v. John Deere Co., 383 U.S. 1 (1966). To guard against the often subtle effects of such hindsight bias, the Federal Circuit has long used a "teaching, suggestion, or motivation" ("TSM") test, under which an invention will not be deemed obvious unless the prior art as a whole contained at least an implicit suggestion to combine the relevant prior art references. See, e.g., KSR Int'l. Co. v. Teleflex, Inc., 550 U.S. 398 (2007); Dystar Textilfarben v. C.H. Patrick, 464 F.3d 1346 (Fed. Cir. 2006); In re Kahn, 441 F.3d 977 (Fed. Cir. 2006).

While objecting to "the way the Federal Circuit applied its TSM test" to a particular set of facts, the Supreme Court has held that "[t]here is no necessary inconsistency between the [TSM] test and the Graham analysis." KSR, 550 U.S. 398 (2007). The Court affirmed that "[t]he TSM test captures a helpful insight: A patent composed of several elements is not proved obvious merely by demonstrating that each element was, independently, known in the prior art." Id. Rather, it is necessary

"to determine whether there was an apparent reason to combine the known elements in the fashion claimed by the patent at issue." Id. Further, the Court held that "[t]o facilitate review, this analysis should be made explicit." Id. (citing In re Kahn, 441 F.3d 977, 988 ("Rejections on obviousness grounds cannot be sustained by mere conclusory statements; instead, there must be some articulated reasoning with some rational underpinning to support the legal conclusion of obviousness.")).

Applicant submits that without Applicant's disclosure as well impermissible hindsight bias, there is no motivation or apparent reason to make the proposed combination and modification of Moore to attempt to achieve the invention of claim 1. That is, taken as a whole, the prior art fails to provide any implicit reason to make the proposed combination. Moore is directed to a utility for interrupting event processing in server-side programs executing within a virtual machine. See, e.g., Moore, Abstract; Background; Figure 2; column 3, line 45-column 7, line 64. Nitz is directed to systems and methods for facilitating communication between sub-applications that have been combined to form a larger, vertical application. See, e.g., Nitz, Abstract; Background; column 4, line 23-column 5, line 8. Given the divergent subject matter disclosed in the two references, it is unclear why one skilled in these arts would look to these references for the purpose making the combination suggested by the Examiner.

Further, in order to properly combine references to support an obviousness rejection, an Examiner is required to articulate an apparent reason for the combination. See, e.g., KSR, 550 U.S. 398 (2007). In the present application, the Examiner identified a reason to combine Moore and Nitz as "it allows for the selection of the correct task to receive an event that it is intended for." This conclusory statement provides absolutely no guidance regarding how the combination of these two particular references would have been obvious to one of ordinary skill in the art. Indeed, there is simply no reason, other than the teachings of the present application, to make the proposed combination.

It is also unclear how Moore and Nitz could be combined without modifying or destroying the intended function of one or both of the references. It is well-known that a proposed modification cannot render the prior art unsatisfactory or change the principle of operation of a reference. See MPEP § 2143.01. As noted, Moore involves

interrupting event processing in server-side programs of a virtual machine such that when a subsequent external event impacts the processing related to a previous external event, the processing of internal VM events relating to the previous external event may be halted or altered as appropriate. In contrast, Nitz employs broker and agent elements to translate and pass messages between combined computer applications and does not involve event processing or virtual machines in any respect. It is therefore unclear how the specifically defined message passing techniques of Nitz could be employed to receive, dispatch, and process external and/or internal VM events within the Moore system. The Examiner proposes to combine Moore and Nitz in a manner that is contrary to common sense, contrary to the teachings of the cited patents, and likely to destroy the functionality of one or both references.

In summary, the combination of Moore and Nitz fails to teach or suggest each and every element required by claim 1. Neither is there any teaching, suggestion, or motivation, nor any rational underpinning, for the proposed combination of Moore and Nitz. Accordingly, Applicant submits that claim 1 is allowable as presented and requests the rejection be withdrawn. Likewise, Applicant submits that because dependant claims 2-6 and 8-11 depend from claim 1, they are also allowable.

Independent Claim 12

As discussed above, independent claim 12 stands rejected as being unpatentable over the combination of Moore and Nitz. Applicant respectfully requests reconsideration because the proposed combination of Moore and Nitz, even if proper, does not teach each element presented by the combination of features required by claim 12 to one of ordinary skill in the art.

Claim 12 is directed to a computer-implemented virtual machine for processing platform-specific events associated with a first platform, where the virtual machine concurrently supports first and second tasks. Specifically, claim 12 requires first and second tasks concurrently operating on the virtual machine and a platform-specific event dispatcher that operates to (1) simultaneously receive a plurality of platform-specific events associated with the first platform, where each platform-specific event is an external event initiated externally to the virtual machine, and (2) concurrently select,

for each of the platform-specific events, a selected task that defines which of the first and second tasks should receive the platform-specific event. Though claim 12 is written in apparatus form, it includes limitations similar to those of claim 1, and therefore, the reasons for allowing claim 1 are equally applicable to claim 12.

Additionally, claim 12 has been amended to include limitations similar to those previously included in claim 14, and claim 14 has been canceled. Specifically, independent claim 12 further requires a platform-specific event-repository for the selected task and a platform-specific event handler for the selected task. The platform-specific repository provides event storage prior to processing by the selected task, and the platform-specific event-dispatcher operates to place the platform-specific event in the platform-specific event-repository and invoke the platform-specific event-handler to initiate processing of the platform-specific event.

As explained in Paragraphs [0032]-[0035] and shown in Figure 2 of the original specification, one embodiment of a virtual machine 204 may include a smart event-dispatcher (i.e., a manager) 212. The smart event-dispatcher 212 may dispatch a specific-platform event E1 to a task 214 and a specific-platform event E2 to a concurrently supported task 216. In greater detail, the smart event-dispatcher 212 may place the platform-specific event E2 in an event-repository 220 (e.g., event storage with a first-in, first-out queue) and notify an event-handler 221 associated with the task 216 that an external, platform-specific event has been delivered. This invokes of the event-handler 221 associated with the task 216 and initiates processing of event E2 on behalf of the task 216.

The Examiner cites column 5, lines 8-22 of Moore as teaching an event-repository. However, a review of Moore reveals that Moore discloses an event dispatcher 160 that maintains a cross-reference listing of executing internal VM events and associated event monitors to maintain a proper relationship between VM events and event monitors. Thus, Moore does not disclose an event-repository that serves as event storage for events that await processing with the selected task. Moreover, Moore only maintains the cross-reference listing with reference to internal VM events, which, as discussed above, are distinguishable from external events initiated outside the virtual

machine. Thus, Moore fails to teach an event-repository having the functionality required of claim 12.

The Examiner further cites column 4, lines 43-44 of More as teaching an event-handler for the selected task, and column 4, lines 64-67, column 5, lines 2-7 and 19-22, and column 6, lines 34-35 as teaching (1) placing the platform-specific event in the event-repository, (2) invoking the event-handler to initiate processing of the platform-specific event. Again, the so-called "event-repository" of Moore is merely a cross-reference list and does not serve as event storage within which an event may be placed. Further, each passage that the Examiner cites refers to dispatching and processing internal VM events, not external events as required by claim 12. Whereas claim 12 involves the processing of simultaneously received, external platform-specific events, Moore involves the processing of internal VM events that are initiated within the virtual machine in response to the consecutive receipt of external events.

For these reasons as well as those provided with respect to claim 1, discussed above, Applicant believes that independent claim 12 is patentable over the cited references and requests the rejection be withdrawn. Likewise, because claim 13 depends directly from claim 12, Applicant believes that it is also allowable.

Independent Claim 20

As discussed above, independent claim 20 stands rejected as being unpatentable over the combination of Moore and Nitz. Claim 20 is directed to a computer readable storage medium including a computer program for processing platform-specific events by a virtual machine that operates on a first platform. The virtual machine concurrently supports first and second tasks. Though claim 20 is written in apparatus form, it includes limitations similar to those of claim 1, and therefore, the reasons for allowing claim 1 are equally applicable to claim 20. Thus, Applicant respectfully requests reconsideration because the proposed combination of Moore and Nitz, even if proper, does not teach each element presented by the combination of features required by claim 20 to one of ordinary skill in the art. Further, Applicant submits that the combination of Moore and Nitz is improper. Thus, Applicant believes that claim 20 is patentable over the cited references and asks that the rejection

be withdrawn and the claim allowed. Because claim 22 depends directly from claim 20, Applicant believes that it is also allowable and asks that it be allowed.

Claim Rejections Under 35 U.S.C. § 103 Based on Moore, Nitz, and Gershman

Claims 8-11 and 15-19 stand rejected under 35 U.S.C. § 103(a) as being unpatentable over Moore in view of Nitz, and further in view of U.S. Patent No. 6,199,099 ("Gershman").

As discussed above, claims 8-11 depend (directly or indirectly) from claim 1 and are believed to be allowable for at least the reasons provided for claim 1. Claims 15-19 have been canceled.

Conclusions

The references of record and not relied upon by the Examiner have been considered and all pending claims are believed to be patentable over those references, as well as the references relied upon by the Examiner to reject claims.

Based upon the foregoing, Applicant believes that all pending claims are in condition for allowance and such disposition is respectfully requested. In the event that a telephone conversation would further prosecution and/or expedite allowance, the Examiner is invited to contact the undersigned.

Please credit any overpayment or charge any underpayment to Deposit Account No. 50-1419.

Respectfully submitted,

MARSH FISCHMANN & BREYFOGLE LLP

Date: 3/12/10

By: Libby Huskey

Libby A. Huskey
Reg. No. 59,087
720-562-5507 Tel